Pergamon

Computers Ops Res. Vol. 21, No. 8, pp. 927–940, 1994

0305-0548/94 $7.00 + 0.00

A GENETIC ALGORITHM FOR THE TALENT SCHEDULING PROBLEM

ANNA-LENA NORDSTRÖM[1,†] and SULEYMAN TUFEKCI[2,‡]

[1]The Graduate School of the Royal Institute of Technology, Stockholm, Sweden and [2]Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, U.S.A.

Scope and Purpose—This paper presents an interesting talent scheduling problem, and provides an efficient hybrid genetic algorithm for solving it.

It is well known that many simple applications of genetic algorithm to combinatorial optimization problems, such as the traveling salesman problem, prematurely converge to a mediocre local optima. In this paper we demonstrate that a similar problem exists in the solution of the talent scheduling problem as well. In overcoming this difficulty we have developed several hybrid algorithms which use limited pairwise interchange procedure within the simple genetic algorithm framework.

This paper shows that intelligent imbedding of other heuristic procedures within a genetic algorithm framework may result in very efficient algorithms to solve many NP-hard problems. This paper also indicates that the concept of hybrid genetic algorithms may also be used for solving clustering and part-family formation problems in Group Technology.

Abstract—We present the talent scheduling problem and provide several hybrid genetic search algorithms for solving this problem. The problem is of scheduling the independent activities of a project (such as a movie) so that the cost of keeping talents (actors) on the project site when not needed is minimized. The problem is an NP-hard problem. The proposed hybrid algorithms use limited pairwise interchange heuristic procedure within the simple genetic algorithm framework. We report very impressive results on a set of problems with known optimal solutions.

1. INTRODUCTION

The talent scheduling problem (tsp) arises in shooting of movies on locations, scheduled maintenance of a remote facility by experts, or in turn key factory constructions and automations. In all these applications some talents (expert repairmen, movie stars, design engineers etc.) are needed in remote locations to perform certain tasks. Not all talents are needed at all times during the execution of a project. Each of those who are required to be present at the remote site are paid perdiem allowance. In addition, each talent's travel cost is also paid by the company who is in charge of the project. If the travel cost is not a significant portion of the problem, then it may be optimal to send the talents of the site in a just-in-time fashion, and return them to the base for the days when their services are not needed. Usually, the travel of talents requires the expensive mode of transportation, such as first class commercial airlines or chartered flights. In such cases, the just-in-time rule may not be economically feasible. Furthermore, it may also be impractical from the human factors point of view. Such a solution will make talents spend a great deal of their time in tiring and boring travels. There is also the issue of unexpected delays in just-in-time arrivals due to many transportation related difficulties.

Because of the inherent difficulties involved in multiple visits to project sites as presented above, many companies prefer to keep the talents on site from the first day until the last day in which they are needed. Therefore, the talents get paid perdiem not only on those days whose services are required but also on those days when their services are not needed. A day in which the service of

†Anna-Lena Nordström received her masters' degree from the Royal Academy of Sweden. She is currently employed in private business.

‡Suleyman Tufekci received his B.A. and M.S. in Industrial Engineering at the Middle East Technical University of Ankara, Turkey. Currently, he is an associate professor at the University of Florida in Gainesville. His current research interests include developing decision support systems, network modeling of emergency evacuation problems, development of emulated flexible manufacturing facilities, and optimization.

a talent is not needed, although the talent is present at the project site, is called a "hold day" for that talent.

The problem differs from an ordinary project scheduling problem in one important aspect. The activities in the tsp are assumed to be independent, whereas in ordinary project scheduling activities are related to each other through precedence constraints. Film production is certainly one project type whose activities may be assumed to be independent. In the production of a feature film, the shooting sequence is not necessarily the same sequence as we see in the final version. It is plausible, that some scenes in the production of a movie or some activities in a maintenance project may have partial precedence relationships. If such partial precedence relationships are present in the problem, we need to combine those related activities into one aggregate activity. This process eliminates all partial precedence relationships from the problem. The resultant aggregate activities are now independent.

The talent scheduling problem is introduced by Cheng *et al.* [1]. In their work, the problem is posed as a combinatorial optimization problem. The objective function used is one of minimizing the total cost of hold days paid to the talents. They show that the problem is NP-hard by proving that the optimal linear arrangement problem [2] can be reduced to an instance of tsp in polynomial time.

Cheng *et al.* [1] point out that this problem has not been studied in the literature. Their research indicate that all the work related to this problem in the movie industry is of qualitative in nature and not well documented. They propose a branch and bound procedure and a heuristic procedure to solve this problem. They report that the branch and bound algorithm can solve problems up to 14 talents and 14 days within an acceptable computer time. For problems beyond 14 days and 14 talents, they claim the proposed heuristic performs quite successfully in reasonable computer times. However, this claim is not substantiated in that paper.

We have performed an extensive survey in scheduling literature with the hope of locating work in talent scheduling or related areas. We have not found any immediately related literature. However, we have discovered several papers which deal with clustering of objects into groups. King and Nakornchai [3] proposed a rank-order cluster algorithm in forming part families in group technology area. Later, Chan and Millner [4], Askin and Subramanian [5], Chandrasekharan and Rajagopalan [6], Steudel and Ballakur [7], Khator and Irani [8], and Vanelli *et al.* [9] have proposed similar clustering algorithms for the part family formation problem. The proposed algorithms consider a binary incidence matrix. Each column of this matrix represents a component or part to be processed and each row corresponds to an operation. The objective of all these cluster algorithms is to bring together those components that need the same or similar set of operations in clusters. Each closely packed cluster is then assumed to form a parts family which is used for forming manufacturing cells.

Although the aim in all these clustering algorithms is to manipulate rows and columns of a binary matrix to form a block diagonal matrix, they are not useful for talent scheduling problem. The primary concern in all these clustering algorithms is to produce nonoverlapping submatrices from the original binary matrix. However, within each submatrix the location of ones and zeroes are insignificant for the clustering problem but extremely important for the talent scheduling problem. Furthermore, a large group of these algorithms do not consider cost explicitly.

There has also been some work on clustering the nodes of a given graph in the literature [10]. In graph clustering the objective is to determine the level and location of each vertex in the hierarchical graph so that the number of edges-crossings is minimized. Once again, these algorithms lack the necessary ingredient the cost objective function to be useful for the talent scheduling problems.

Traveling salesman problem is one of the most celebrated problems in the operations research literature. This problem has also been tackled by genetic algorithm [11–15]. The talent scheduling problem can be viewed as finding the minimum cost Hamiltonaian path among the first n integers, where n is the number of days to schedule. However, in all previous work on similar problems, including the traveling salesman and the Hamiltonian path problems, the arc cost associated with a pair of city is fixed and does not change. On the other hand, this quantity is the function of the location of the other cities in a given tour in talent scheduling problem. It is this property that makes the talent scheduling problem a challenging problem to tackle.

In this paper we present several variations of a hybrid genetic algorithm using PMX crossing

procedure of Goldberg [12]. The interchange heuristic procedure is integrated into the algorithm to solve the tsp. Our test results show that the proposed hybrid heuristic is extremely efficient, and finds optimal solutions or very near optimal solutions to some randomly generated problems with known optimal solutions. On other randomly generated problems, the proposed genetic algorithm outperforms Cheng *et al.* heuristic both in quality of the results and in computational times. We present the talent scheduling problem in Section 2. Genetic algorithms are briefly discussed in Section 3. The simple genetic algorithm and hybrid algorithms with interchange procedure are introduced in Section 4. The results of our experiments are documented in Section 5. Section 6 concluces our work.

2. TALENT SCHEDULING PROBLEM

In presenting the talent scheduling problem we will use a film production example as the project. Assume that a complete film production project is divided into n independent shooting days (scenes) and there are m talents (actors) involved at different days of the movie (project). In a film production problem, the independent activities are the shooting days (or scenes) and talents are the movie stars needed in each of these days.

An $(m \times n)$ binary matrix T, called the "day out of days matrix" is defined [1]. The element, T_{ij}, of this matrix is set to 1 if talent i is needed in the project, for scene j. The cell is zero otherwise. We also have an m-vector, $\mathbf{c}$, whose component c_i correspond to the per-diem payment for talent i. Let $\mathbf{s}$ be any permutation of integers $\{1, 2, \ldots, n\}$. Also let $T(\mathbf{s})$ represent the T matrix whose columns are permuted according to the permutation sequence $\mathbf{s}$. We can express the relationship as

$$T_{ij}(\mathbf{s}) = T_{is(j)}, \quad \text{for } i = 1, \ldots, m, j = 1, \ldots, n. \tag{1}$$

Here, $s(j)$ is the scene that is scheduled to be filmed in day j of shooting.

It is clear that any permutation sequence $\mathbf{s}$ determines a project shooting sequence. We denote with $l_i(\mathbf{s})$ and $e_i(\mathbf{s})$, the earliest day and the latest day a talent has to be present for the project, dictated by the permutation sequence $\mathbf{s}$. It is easy to see that the talent will stay exactly $(l_i(\mathbf{s}) - e_i(\mathbf{s}) + 1)$ days on location. If r_i is the actual number of shooting days a talent i is needed on the project, then the number of hold days for talent i is given by

$$h_i(\mathbf{s}) = (l_i(\mathbf{s}) - e_i(\mathbf{s}) + 1) - r_i. \tag{2}$$

We note here that, $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ can easily be calculated by $\mathbf{r} = T\mathbf{e}$, where $\mathbf{e}$ is an $(n \times 1)$ column vector with all ones.

Thus, the total cost of hold days is given by

$$z(\mathbf{s}) = \mathbf{c}\mathbf{h}^T(\mathbf{s}), \tag{3}$$

where, $\mathbf{h}(\mathbf{s}) = (h_1(\mathbf{s}), \ldots, h_m(\mathbf{s}))$.

We now present the combinatorial optimization model for the talent scheduling problem.

$$\text{minimize} \quad z(\mathbf{s}) = \mathbf{c}\mathbf{h}^T(\mathbf{s}) \tag{4}$$

$$\text{subject to} \quad \mathbf{s} \in \mathbf{S}. \tag{5}$$

Here $\mathbf{S}$ represents the set of all permutations of integer numbers (1, ..., n).

Cheng *et al.* prove that the talent scheduling problem is NP-hard. The proof follows after the optimal linear arrangement problem is reduced to a recognition version of the talent scheduling problem [2]. They propose a branch and bound algorithm to solve this problem. Each node k of the branch and bound tree corresponds to a partial schedule P_k. The procedure constructs the schedule from outside in. First $s(1)$ gets determined followed by $s(n)$. This process continues on $s(2)$, $s(n-1)$, $s(3)$, and so on, in that order.

There is some amount of similarity between the traveling salesman problem (TSP) and the talent scheduling problem (tsp). In both problems, the solution set is the permutation of first n positive integer numbers. However, in TSP, the cost between city i and city j is fixed and is not a function of the permutation. In contrast, the number of hold days for a talent between a shooting day i and a shooting day j, depends not only which scenes are scheduled in those days, but also when the other scenes are scheduled in the project.

In finding a good starting solution for a good upper bound, Cheng *et al.* propose an outside in heuristic procedure (CDL heuristic hereinafter). This heuristic procedure allocates the scenes to the shooting days from outside-in as described above. At each stage (iteration) of the heuristic, a scene which will add the smallest increase to the lower bound at that node is placed in the partial schedule. The heuristic starts by considering $n(n-1)/2$ pairs of days as potential first and lst day of the partial schedule. Once the heuristic provides a complete permutation schedule, a pairwise interchange procedure is applied to this schedule. In this part any two scenes (shooting days in **s**) are considered for an interchange (swap). If such a swap reduces the total cost of hold days, then the interchange takes place. Otherwise another pair is tested for interchange. This process of swapping the shooting days continues until no more pairs can be swapped to reduce the total hold cost.

The proposed CDL heuristic is a pseudopolynomial algorithm with time bound of

$$O(mn)O(mn^2(n-1)c_{\max}) = O(m^2n^4c_{\max}).$$

Here, $c_{\max}$ represents the maximum entry in the **c** vector.

Cheng *et al.* [1] claim that this heuristic algorithm produces very good results in reasonable computer time. However, their claim is not substantiated by experimental results for problems with more than 14 days and 14 talents. As expected, the branch and bound procedure requires fairly large CPU times for problems with 14 or more talents and shooting days. Although the CDL heuristic procedure yields fairly good results on the problems tested, its CPU time requirement will grow several orders of magnitude with the growing size of the working days.

To alleviate this CPU time problem, and to achieve better results than the one provided by the CDL heuristic, we decided to develop a genetic algorithm for this problem. The next section presents a general introduction to genetic algorithms. Our proposed simple and hybrid genetic algorithms for the tsp problem are presented in Section 4.

3. GENETIC ALGORITHMS

Holland [16] describes genetic algorithms (GA) as search algorithms based on the mechanics of nature's selection and genetic rules. Genetic algorithms search through large spaces quickly in an attempt to find the global optima. In contrast to many other techniques, the approach depends upon a whole generation of different solutions instead of a single point.

The GAs work as follows: feasible solutions, represented as strings are generated randomly and assigned a weight or a fitness value by an appropriate fitness function. The fitness function may be, for example, the objective function. These available solution strings are then selected with a probability proportional to their fitness value and crossed in a predetermined way. Each new string created this way contains ordering information partially determined by each of its parents. This process of crossing continues repeatedly during the course of the algorithm, as long as new strings with better fitness are obtained and the predetermined time limit in the execution time is not exceeded.

A generic algorithm basically consists of three operators. These operators are called reproduction, crossover and mutation. In what follows we describe each of these operators.

3.1. Reproduction

This operator selects a number of sample strings from the current population for crossing depending on their fitness. This is done by constructing a Monte Carlo wheel and giving each string a space on this wheel proportional to its fitness. The probability of a string to bc selectged for crossing is then given by the ratio of the fitness of an individual string to the total fitness of the entire population. The samples are chosen by spinning the wheel.

It is common knowledge, that early in the evaluation, the population is dominated by strings with low fitness function values. There are only a few extraordinary strings with fitness values relatively very high with respect to other members of the population. This situation may cause what is known as premature convergence to a local optimum. Towards the end of the algorithm the situation becomes just the opposite. The fitness of many best strings are closer to the average fitness. Even if there are a few significantly higher fitness values, the ones around the average fitness are more numerous. This causes a majority of the wheel slot to be allocated to mediocre strings.

In order to overcome this problem, Goldbert [1] suggests a linear scaling of the fitness function,

depending on where the average lies, relative to maximum and minimum fitness values in a given population. He suggests a linear scaling of the fitness values in such a way that the average raw fitness is equal to the average scaled fitness. This results in the following two relationships:

$$f_{\text{scaled}} = a \cdot f_{\text{raw}} + b \tag{6}$$

$$f_{\text{scaled}}(\max) = c_{\text{mult}} \cdot f_{\text{raw}}(\text{aver}). \tag{7}$$

Here, a and b are two coefficients that can be chosen in several different ways and c_{mult} is the expected number of desired copies for the best population member. $c_{\text{mult}} = 1.2$–2 has been suggested in the literature for population sizes of 50–100. In this study, we use $c_{\text{mult}} = 2$, $a = f_{\text{raw}}(\text{aver})/D$, and $b = f_{\text{raw}}(\text{aver})[f_{\text{raw}}(\max) - 2f_{\text{raw}}(\text{aver})]/D$, where $D = f_{\text{raw}}(\max) - f_{\text{raw}}(\min)$. Here, $f_{\text{scaled}}(\max)$, $f_{\text{raw}}(\max)$ and $f_{\text{raw}}(\min)$ represent the maximum scaled score, maximum raw score and the minimum raw score, respectively.

3.2. Crossover

The crossover is performed by randomly selecting two mates (strings) from the current pool of strings (population) and crossing them with a specified probability. One of the simplest ways of achieving this crossover is by partially exchanging sections of strings between two random points. The old population is discarded if the mating produces successful offsprings. Another version is the incremental version, where the new strings are included in the old population as the crossing proceeds.

It is shown by DeJong [17] that a good GA performance requires the choice of a high crossover probability, a low mutation probability and a moderate population size. Commonly used crossover rates range between 0.6 and 0.95. In this study, after experimenting with several crossover rates in this range, we have adopted the value 0.7 for all further testings.

3.3. Mutation

The mutation is performed by random alteration of a bit in a solution string with a specified mutation probability. In a binary string this simply means changing a 1 to a 0, or vice versa. DeJong [17] suggested a mutation rate of approximately 1/(population size). However, mutation only plays a secondary role in the success of genetic algorithms.

Any genetic algorithm to be designed must obey the fundamental theorem of genetic algorithms which states that "*Schema, or substrings, which are short, of low order and above average will be expected to survive.*"

The order of a substring is the number of fixed positions within the substring. The length is the distance between the first and last specific position in a substring. Above average is simply a substring with fitness above average compared to the rest of the population. A good subtour in a traveling salesman problem may represent a substring that fulfills the requirements of this theorem. Similarly, in the tsp, a good subsequence of days scheduled will also fulfill this requirement.

In all our implementations we have performed the mutation by simply interchanging the position of two shooting days selected randomly with the accepted mutation probability.

Since there is a good deal of similarity between the talent scheduling problem and the traveling salesman problem, our genetic algorithm will borrow some techniques from the applications of GAs on traveling salesman problems [11–15].

The traveling salesman problem (TSP) is generally formulated as a minimization problem. In order to easily implement the Monte Carlo selection procedure during the implementation of genetic algorithms, it is customary to transform the problem into a maximization problem. The classic and commonly used cost-to-fitness transformation is given by

$$f(x) = C_{\max} - g(x)$$

In this, $g(x)$ is the individual fitness value of a solution x in the current population and $C_{\max}$ is chosen to be the highest tour cost in the present population. This has the effect that the string with the highest tour cost vanishes since it will have no space in the wheelslot.

The representation of a TSP tour as a permutation of n integers is not consistent with the crossover operator. Usually tours are not preserved after crossover. One modification of the crossover

operator is suggested by Goldberg and Lingle [13], called Partially Matched Crossover (PMX). To illustrate PMX we draw on Goldberg's [12] example:

Consider two strings where the two crossing sites have been picked randomly:

$$A = 9\ 8\ 4|5\ 6\ 7|1\ 3\ 2\ 10$$

$$B = 8\ 7\ 1|2\ 3\ 10|9\ 5\ 4\ 6$$

PMX proceeds by positionwise exchanges. First, mapping string B to string A, the elements 5, 6 and 7 in string A exchange places with elements 2, 3 and 10 in string A, respectively. Next, string A is mapped to string B. Elements 2, 3 and 10 in string B are replaced by elements 5, 6 and 7 in string B, respectively. As the result of this PMX we now have two offspring, A' and B' where

$$A' = 9\ 8\ 4\ 2\ 3\ 10\ 1\ 6\ 5\ 7$$

$$B' = 8\ 10\ 1\ 5\ 6\ 7\ 9\ 2\ 4\ 3.$$

The representation of the TSP tour as a string of city numbers makes the mutation operation very easy to implement. Two cities are randomly picked and swapped with each other obeying the probability of mutation.

4. GENETIC ALGORITHMS FOR THE TALENT SCHEDULING PROBLEM

At the beginning of our development process, we have designed and applied the Simple Genetic Algorithm to the talent scheduling problem as described below. After analysing the initial results we have concluded that the SGA was not competitive enough vis a vis the CDL heuristic procedure provided in [4]. We then introduced the next three variations. Each variation basically combines a limited pairwise interchange procedure with the simple genetic algorithm SGA.

4.1. Simple genetic algorithm (SGA)

This is the method as described in Section 3 for the traveling salesman problem. The objective function is the sum of hold days costs over all talents. Since the objective is the minimization of the total cost of hold days, we use the cost to fitness transformation as described in Section 3.3 to convert it to a maximization problem. For each tour (string) $\mathbf{s}$ in the population, the term $f(\mathbf{s}) = C_{\max} - g(\mathbf{s})$ represents the fitness function for each string where $g(\mathbf{s}) = \mathbf{ch}^{\mathrm{T}}(\mathbf{s})$, as defined in Section 2. In all our experiments, involving this version we used crossover rates of 0.6, 0.7, or 0.8, a mutation rate of 1/(population size), and population sizes of 20, 50 and 100. In scaling the fitness values, we used Goldberg's [12] linear scaling procedure as given in Section 3.1. In the scaling formulas the c_{mult} used was 2.0. We have determined and used the values of a and b, as given in Section 3.1.

The simple genetic algorithm (SGA) starts by generating a population of size $L(=20$, or 50 or

Table 1. Anatomy of the SGA

EVALUATION FUNCTION:
$C_{\max} - g(\mathbf{s}) = C_{\max} - \mathbf{ch}^{\mathrm{T}}(\mathbf{s})$
POPULATION
Representation: permutation of first n integers
Initialization Technique: random assignment
Reproduction Technique: generational replacement
Parent Selection Technique: Monte-Carlo roulette
Fitness Technique: Goldberg's [12] linear cost-to-fitness transformation
Population Sizes: 20, 50 and 100
Number of Iterations: 50 and 100
REPRODUCTION
Reproduction Operator: PMX procedure of Goldberg and Lingle [13]
Mutation Rate: 1/population size
Crossover Rates: 0.6, 0.7 and 0.8

100), randomly. A Monte Carlo wheel is designed, based on the fitness value of each string. The amount of space allocated to each string is determined by the ratio of each individual string's scaled fitness value to the total scaled fitness value of the entire population. The wheel is randomly spinned for selecting the pairs of strings for crossings. Those selected pairs are then subjected to crossover procedure with a designated acceptance probability (0.6, 0.7 or 0.8). The mutation is introduced as simply swapping two cities in a string with the probability of (1/population size). We have used PMX as the crossover procedure. The algorithm terminates when either the strings converge to a local optima or the prescribed iteration count is reached. Table 1 summarizes the anatomy of SGA in a format similar to the format used in Davis [11].

4.2. Pairwise interchanges on the initial population (SGAI)

This version starts exactly the same as the SGA. However, before it proceeds with the crossing and mutation on the randomly generated initial population, a limited one-time pairwise interchange procedure is applied to each string in the population. The pairwise interchange works as follows.

PROCEDURE INTERCHANGE
INPUT: The population size, K, members of current population $\mathbf{s}_k$, $k=1, 2, \ldots, K$, $\mathbf{c}$, m, n and T
BEGIN
    FOR $k=1$ TO K DO
    FOR $i=1$ TO $n-1$ DO
    FOR $j=2$ TO n DO
        Swap days i and j in $\mathbf{s}_k$ if it yields a lower total hold day cost.
    ENDFOR $\{i, j, k\}$
END.

Note that the additional computational burden introduced by this version is of $O(Kn^3m)$, since each swap check requires $O(mn)$ effort ($O(n)$ effort to determine the hold day cost for each talent for each one of m talents) and there are $O(Kn^2)$ such swap checks required in the above procedure. After the initial population is obtained by this method, the SGAI continues exactly as the SGA, as given in Section 4.1. Note that the above procedure is deterministic. There will be exactly $K(n-1)^2$ attempts of swapping before the procedure terminates.

4.3. Pairwise interchanges on the final population (SGAF)

In this version the SGA is applied until the algorithm terminates. On the final population obtained, we apply the procedure INTERHCHANGE as described above to all distinct strings in the final population. The computational burden of this pairwise interchange is also $O(Kn^3m)$.

4.4. Pairwise interchanges in all populations (SGAA)

In this version we alternate between the SGA and the procedure INTERCHANGE. The starting population is randomly generated and pairwise interchange is then performed upon this population, exactly as in SGAI. At each iteration, after a complete set of crossovers and mutations are performed on a given population, pairwise interchange is applied to the resultant population before the next round of crossovers and mutations start. Note that the additional computational burden of SGAA over SGA is of $O(qKn^3m)$, where q is the number of iterations allowed in the genetic algorithm SGA.

Therefore, for fixed population size and number of iterations, all of the proposed heuristics above are of polynomial in the size of the problem.

In evaluating the fitness function of each newly generated string, and eventually the overall fitness of a given population, we have to compute the hold cost for each talent for two different cases. One is when a mutation or pairwise interchange modifies a string and the other one is when the crossover procedure creates a new string. We have used the following fact in evaluating the incremental effect of a pairwise interchange on the total fitness function. Note that given a feasible schedule $\mathbf{s}$, if two arbitrary days $j(\mathbf{s})$ and $k(\mathbf{s})$ on this schedule are interchanged, the incremental contribution of talen i to the total fitness function due to this interchange is zero if $T_{ij(\mathbf{s})}(\mathbf{s}) = T_{ik(\mathbf{s})}(\mathbf{s}) = 0$, or $T_{ij(\mathbf{s})}(\mathbf{s}) = T_{ik(\mathbf{s})}(\mathbf{s}) = 1$, or $e_i(\mathbf{s}) < j(\mathbf{s})$, $k(\mathbf{s}) < l_i(\mathbf{s})$, where $e_i(\mathbf{s})$ and $l_i(\mathbf{s})$ represent the earliest and the latest shooting days for talent i with respect to a feasible schedule $\mathbf{s}$. By using this fact we have eliminated a significant amount of computational burden from our algorithms. In the implementation of the

algorithms, we simply keep track of $\mathbf{e}_i(\mathbf{s})$ and $l_i(\mathbf{s})$ for each talent i and update them each time a pairwise interchanges takes place.

5. EXPERIMENTAL RESULTS ON THE TALENT SCHEDULING PROBLEM

All our algorithms were coded in Turbo PASCAL and ran on a 16 MHz, 80386 based and 25 MHz, 80486 based personal computers. The CPU times reported in this paper exclude the data input and output times to the program. We have clearly identified the experiments that used 16 MHz machine and the experiments that used the 25 MHz machine.

We have tested the SGA as described in Section 4, extensively on four initial test problems ($(m=6, n=10)$, $(m=10, n=20)$, $(m=19, n=28)$, $(m=25, n=25)$) generated randomly with different T matrix density. On those test problems, we have tried three crossover rates (0.6, 0.7 and 0.8), three different population sizes (20, 50 and 100), and two different choice on the number of iterations (50 and 100). The results were not very promising. The strings prematurely and consistently converged to a mediocre local optimum as compared to the CDL heuristic.

In light of these initial experimentations, we decided to implement the CDL heuristic, SGA, SGAI, SGAF and SGAA as described in Section 4 on the 28-day test problem provided by Cheng *et al.* [1]. The results of this experiment is given in Table 2. After analysing the results of this experiment, we have decided to test the SGA, SGAI and SGAF on other randomly generated problems. In all those experiments we have conducted with different population sizes, number of iterations, and crossover rates the SGA, SGAI and SGAF have consistently performed inferior to the CDL algorithm. However, the SGAA have performed better than the CDL heuristic on these problems. At this point we decided to focus our attention on the SGAA against the CDL heuristic in the remaining experimentations.

Table 2. The CDL heuristic, SGA, SGAI, SGAF, and SGAA tested on the 28-day 8-talent problem

	CLD heuristic	SGA	SGAI	SGAF	SGAA
Objective function	18,100	31,000	19,000	26,500	15,600
CPU time (s)	3.49	3.18	1.57	3.39	1.43

The first set of problems are generated randomly. First a density for the T matrix is defined. The nonzero entries of the T matrix are then generated randomly until the desired density is reached. At this point the T matrix is checked for rows with all zero entries. If none is found the generation of the T matrix is declared successful. Otherwise, those rows with all zero entries are randomly inserted with a single nonzero entry. We have generated three test problems: one with 10 actors and 40 days with 25% density and two with 10 actors and 50 days with 15% density. We have used population sizes of 50 and 100, in these experiments. Each problem is solved four times by the CDL heuristic, the SGAA with the population size 50 and the SGAA with the population size 100. To be able to test the CDL heuristic on the same test problem more than one time, we scrambled the columns of the T matrix randomly, each time, before presenting it to the algorithm. Figure 1 summarizes the results of these four replications on each of the three test problems.

Although the SGAA have outperformed the CDL heuristic on all problem types, we had no indication of how close we were to the true optimal solutions.

To test the SGAA and the CDL heuristic we have constructed a set of test problems with known optimal solutions. This is accomplished as follows. Let n be the number of days, and m be the number of talents in the tsp. First, we generate m random integers $z_i \geq 1$, $i=1, 2, \ldots, m$ such that $z_1+z_2+\ldots z_m=n$. The row of the T matrix corresponding to the talent $i=1, 2, \ldots, m$ is then filled with 1's between the column Z_{i-1} and Z_i. The rest of the columns are filled with 0's for this talent. Here, $Z_0=0$, $Z_1=z_1$, $Z_i=Z_{i-1}+z_i$, $i=2, \ldots, m$. Note that with this generation procedure, talents i and $i+1$ share only one day in common. Once this process is completed for all talents, the columns of the T matrix is then scrambled randomly before each testing of the CDL heuristic. Each replication generated this way have the same optimal objective value of zero. Note that a solution where the days are scheduled, either in an increasing or in a decreasing order of first n integers are two optimal solutions for these problems.

m = 10 n = 40	CDL Heuristic	SGAA K = 50, q = 50	SGAA K = 100, q = 50
Objective Function	6,700	5,500	5,500
	10,300	5,500	6,100
	7,500	5,500	5,500
	7,700	5,700	5,500
Avg. CPU (sec)	699	634	1287

m = 10 n = 50	CDL Heuristic	SGAA K = 50, q= 50	SGAA K = 100, q=50
Objective Function	9,500	400	800
	7,300	1,200	1,000
	1,000	800	1,400
	2,900	1,600	1,000
Avg. CPU (sec)	403	380	773

m = 10 n = 50	CDL Heuristic	SGAA K = 50, q = 50	SGAA K = 100, q=50
Objective Function	6,900	9,600	6,900
	12,000	8,100	7,500
	7,100	10,300	8,400
	8,100	10,800	8,100
Avg. CPU (sec)	1590	775	1253

Fig. 1. Test results on three randomly generated problems. K is the population size and q is the number of iterations. Times are reported in CPU seconds.

In studying the results we have discovered that both algorithms found optimal solution for the test problems generated. Studying the test problems generated closely, we realized that the problems generated had great many optimal solutions. The lower bound on the number of alternative optimum solutions is given by

$$N = 2 \prod_{k=1}^{m} (z_k - 1)!.$$

Here, the multiplier 2 comes from the fact that the inverse (mirror image) of each schedule provides a solution of equal objective function value.

To reduce the number of alternative optimal solutions from the randomly generated problems we decided to experiment on a set of scheduling problems where $z_i = 2$ for all $i = 1, \ldots, m$. This automatically implies $n = m + 1$ and $N = 2$ on each problem generated. Following tables in Fig. 2 present the results of our experiments for $m = 10$, $m = 15$, $m = 20$. For each value of m we have replicated the experiment 10 times for four combinations of population size and maximum iteration limit. The tables in Fig. 2 display the worst solution, best solution, the average optimal objective value and the number of times the optimal solution is achieved by each method. In all the problems generated we assumed a **c** vector of all ones. This cost function simply allows us to mininize the number of hold days. Therefore, the optimal objective value is always 0 in all these problems.

As evidenced by the results presented above, on small problems, the SGAA found an optimal solution in all replications. As the problem sizes became large, the SGAA has missed optimal solutions in some replications. However, even in those missed cases, the deviations from the optimal solutions were extremely small. Although the CPU times grew with the size of the problems, this growth was not as pronounced in the SGAA as it was in the CDL heuristic.

To see the effect of problem size in the solution accuracy of these special problems we have finally tested the SGAA and CDL heuristics on a problem with 49 scenes (days) and 48 talents.

m = 10 n = 11	CDL Heuristic	SGAA K=50, q=5	SGAA K=25, q=5	SGAA K=15, q=10	SGAA K=5, q=20
Average Solution	12	0	0	0	0
Worst Solution	22	0	0	0	0
Best Solution	0	0	0	0	0
No. Opt. Found	4	10	10	10	0
Avg. CPU (Sec.)	12	8	7	4	2

m = 15 n = 16	CDL Heuristic	SGAA K=50, q=5	SGAA K=25, q=5	SGAA K=15, q=10	SGAA K=5, q=20
Average Solution	43.1	0	1.4	0.8	3.4
Worst Solution	61	0	6	8	12
Best Solution	0	0	0	0	0
No. Opt. Found	1	10	7	9	5
Avg. CPU (Sec.)	91	40	32	28	11

m = 20 n = 21	CDL Heuristic	SGAA K=50, q=5	SGAA K=25, q=5	SGAA K=15, q=10	SGAA K=5, q=20
Average Solution	4.5	0.9	5.2	3.7	4.0
Worst Solution	12	9	15	6	6
Best Solution	0	0	0	0	0
No. Opt. Found	3	9	5	3	2
Avg. CPU (Sec.)	505	150	80	79	24

Fig. 2. Test results on three randomly generated problems. K is the population size and q is the number of iterations. Times are reported in average CPU seconds. In all these problems $z_i = 2$, $i = 1, \ldots, m$.

m = 48 n = 49	CDL Heuristic	SGAA K = 50, q = 5	SGAA K = 25, q = 5	SGAA K = 15, q = 5	SGAA K = 10, q = 20
Average Solution	71.0	65.0	58.0	74.0	58.2
Worst Solution	92	71	84	80	81
Best Solution	55	31	40	48	45
No. Opt. Found	0	0	0	0	0
Avg. CPU (sec)	9876*	680	331	198	303

* Results of two successful runs. Time limit of 10,000 seconds exceeded on other 2 replications.

Fig. 3. Test results on 48-talent 49-day problems. K is the population size and q is the number of iterations. Times are reported in average CPU seconds. In all these problems $z_i = 2$, $i = 1, \ldots, m$.

We have used 4 pairs of iteration/population size combinations. Each combination is replicated 5 times and minimum, maximum, and average objective function values obtained in all replications together with the CPU times and number of times the optimal was found is listed in the tables of Fig. 3.

The tables in Fig. 3 show that the CDL heuristic deteriorates quickly, both in CPU time and in objective function value, for large values of n and m. Although the SGAA heuristic also deteriorates with the size of the problem, this deterioration was not as severe. The solutions were still in close proximity of the optimal solutions and the CPU times were still affordable.

As a final battery of tests, we deciced to generate 2 additional random problems with (m, n) values of (19, 48) and (43, 48). The number of nonzero entities in the T matrix, for each talent, is generated randomly between 2 to $n/2$ and placed in the proper row, also randomly. We used a several population/iteration combinations for the SGAA heuristic. The results of this experiment is listed in the two tables given in Fig. 4.

m = 19 n = 48	CDL Heuristic	SGAA K = 50, q = 5	SGAA K = 25, q = 5	SGAA K = 15, q = 5	SGAA K = 10, q = 20
Average Solution	301.8	303.5	305.6	300	301.8
Worst Solution	305	307	308	301	307
Best Solution	300	300	304	298	296
Avg. CPU (sec)	1920	450	299	176	427

m = 43 n = 48	CDL Heuristic	SGAA K = 50, q = 5	SGAA K = 25, q = 5	SGAA K = 15, q = 5	SGAA K = 10, q = 20
Average Solution	818	812.8	819.6	821.8	811.6
Worst Solution	824	818	831	830	821
Best Solution	808	808	813	811	808
Avg. CPU (sec)	4381	1273	480	277	870

Fig. 4. Test results randomly generated 19-talent 4-day, and 43-talent 48-day problems.

m =10 n = 40	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	4994.6	4975.2	4974.2
Worst Soln.	5076.0	4992.0	4986.0
Best Soln.	4965.0	4965.0	4965.0
CPU Seconds	210	27	47

m =10 n = 50	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	10474.8	10434.0	10425.0
Worst Soln.	10576.0	10529.0	10601.0
Best Soln.	10381.0	10375.0	10407.0
CPU Seconds	170	42	69

m =10 n = 60	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	14475.2	14477.3	14467.4
Worst Soln.	14742.0	14730.0	14690.0
Best Soln.	14275.0	14285.0	14260.0
CPU Seconds	885	59	99

m =10 n = 70	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	11890.4	12216.4	12157.0
Worst Soln.	12008.0	12501.0	12428.0
Best Soln.	11758.0	12000.0	11812.0
CPU Seconds	1420	79	135

Fig. 5. Results of randomly generated 10-talent problems tested on a 80486 based 25 MHz personal computer.

As expected, the CDL heuristic required much longer CPU time for problems with larger m and n values. The same problems were solved in much shorter CPU time by the SGAA heuristic. Moreover, the solutions obtained by the SGAA were almost consistently better than the CDL results.

To eliminate a possible bias towards any algorithm which may have been caused by the special

m =20 n = 30	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	18104.0	18095.8	18030.6
Worst Soln.	18159.0	18280.0	18185.0
Best Soln.	18022.0	17978.0	17920.0
CPU Seconds	200	30	50

m =20 n = 50	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	32016.0	31634.2	31713.0
Worst Soln.	32393.0	32299.0	32256.0
Best Soln.	31212.0	31152.0	31254.0
CPU Seconds	936	75	124

m =20 n = 60	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	35047.8	36124.0	35044.2
Worst Soln.	35983.0	36845.0	35750.2
Best Soln.	34428.0	35627.0	34450.0
CPU Seconds	1653	95	115

m =20 n = 70	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	44856.6	44920.2	44675.2
Worst Soln.	44965.0	45002.0	44890.0
Best Soln.	44324.0	44600.0	44400.0
CPU Seconds	2649	141	242

Fig. 6. Results of randomly generated 20-talent problems tested on a 80486 based 25 MHz personal computer.

problems with known structure and optimal solutions, we decided to test both SGAA and the CDL heuristic on a battery of randomly generated test problems of varying sizes. These set of problems, however, were solved on a 25 MHz 80486 based personal computer. Each T matrix is generated randomly with overall density of 50%. Lower or higher density rates other than 50% tend to yield many similar solutions probably due to the multiplicity of good solutions and the multiplicity of optimal solutions. Cost for each talent has always been generated between 0 and 200 uniformly. Each m and n combination has been replicated 5 times and the best solution, worst solution, average solution, and the average CPU times are reported in Figs 5–7.

This set of experiments have shown one more time that the proposed hybrid genetic algorithm is robust and almost consistently better than the CDL heuristic in finding better solutions. On the other hand, the CPU time for the CDL heuristic is at least 6 times more than the proposed SGAA. Furthermore the CPU ratios get larger as the problems size gets larger. For example, for the 30 talent 70 day problem, CDL heuristic requires over 20 times more CPU time than the SGAA heuristic. For problems with hundreds of rows and columns the CPU time ratios will continue to get worse for the CDL heuristic.

m =30 n = 40	CDL Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	41830.8	41682.0	41667.6
Worst Soln.	42108.0	41839.0	41735.0
Best Soln.	41537.0	41341.0	41552.0
CPU Seconds	682	71	120

m =30 n = 50	CDL* Heuristic	SGAA K=20, q=5	SGAA K=40, q=4
Avg. Soln.	43321.7	43455.2	43273.6
Worst Soln.	43467.0	44005.0	43873.0
Best Soln.	43172.0	42627.0	43026.0
CPU Seconds	1382	110	187

* results of only three runs.

m =30 n = 60	CDL# Heuristic	SGAA K=20, q=5
Avg. Soln.	52956.5	53137.3
Worst Soln.	53114.0	53600.0
Best Soln.	52400.0	52715.0
CPU Seconds	1900	150

results of three runs

m =30 n = 70	CDL* Heuristic	SGAA K=20, q=5
Avg. Soln.	86779.0	80830.8
Worst Soln.	91819.0	88908.0
Best Soln.	79310.0	78024.0
CPU Seconds	2947	125

* Results of four runs. One run exceeded 3000 second CPU limit.

Fig. 7. Results of randomly generated 30-talent problems tested on a 80486 based 25 MHz personal computer.

6. CONCLUSION

We have presented the talent scheduling problem and proposed a hybrid genetic algorithm to solve this problem. The proposed algorithm SGAA uses the PMX procedure of Goldberg and Lingle [13] for traveling salesman problems in crossing the strings in a current population. To deter the premature convergence and escape from local optima, we introduced pairwise interchange procedure INTERCHANGE to each population before a new set of crossings and mutations are implemented. The computational burden of this exchange procedure on the problem is of $O(qKmn^3)$, where q is the number of iterations, K is the population size, m is the number of talents and n is the number of days in the problem.

The proposed SGAA heuristic compares very favorably with the CDL heuristic on problems with known optimal solutions and also on random problems. The CPU times of SGAA becomes 20 times shorter than the CDL heuristic on large size problems tested. This result was expected because of the computational bounds of the two heuristic presented before. In addition, the SGAA heuristic obtained better solutions than the CDL heuristic almost consistently.

Increasing the population size seems to have improving effect on the quality of the solution in the SGAA heuristic. However, increasing the iteration count did not yield a pronounced

improvement in the quality of the best solution. Generally the best solutions are obtained between the third and the fifth interations for the $K = 20$ test problems. For the $K = 40$ case, the predominant iteration number where the best solution obtained was 3.

A natural extension of this work will be to implement similar genetic algorithms for clustering problems and the traveling salesman problem.

REFERENCES

1. T. C. E. Cheng, J. Diamond and B. M. T. Lin, Optimal scheduling in film production to minimize talent hold cost. Department of Actuarial and Management Sciences, University of Manitoba, Winnipeg, Manitoba, Canada (1991).
2. M. R. Garey, D. S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems. *Theoretical Computer Science* **1**, 237–267 (1976).
3. J. R. King and V. Nakornchai, Machine component group formation in group technology: review and extension. *International Journal of Production Research* **20**, 2, 117–133 (1982).
4. H. M. Chan and D. A. Millner, Direct clustering algorithm for group formation in cellular manufacture. *Journal of Manufacturing Systems* **1**, 1 (1982).
5. R. G. Askin and S. P. Subramanian, A cost based heuristic for group technology configuration. *International Journal of Production Research* **25**, 101 (1987).
6. M. P. Chandrasekharan and R. Rajagopalan, MODROC: an extension of rank order clustering for group technology. *International Journal of Production Research* **24**, 5, 1221–1233 (1986).
7. H. J. Steudel and Al Balakur, A dynamic programming based heuristic for machine grouping in manufacturing cell formation. *Computers and Industrial Engineering* **12**, 215 (1987).
8. S. K. Khator and S. A. Irani, Cell formation in group technology: a new approach. *Computers and Industrial Engineering* **12**, 131 (1987).
9. A. Vanelli and K. R. Kumar, A method of finding minimal bottleneck cells for grouping part-machine families. *International Journal of Production Research* **24**, 2, 387–400 (1986).
10. J. N. Warfield, Crossing theory and hierarchy mapping. *IEEE Transactions SMC* **7**, 7, 505–523 (1977).
11. L. Davis (Editor), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York (1991).
12. D. E. Goldberg, *Genetic Algorithms in Search, Flow and Machine Learning*. Addison-Wesley, New York (1989).
13. D. E. Goldberg and R. Lingle Jr., Alleles, loci, and the traveling salesman problem. *Second Int. Conf. on Genetic Algorithms and Their Applications*, 154–159 (1985).
14. I. M. Oliver, D. J. Smith and J. R. C. Holland, A study of permutation crossover operators on the traveling salesman problem. In *Genetic Algorithms and their Application, Proceedings of the Second International Conference* (Edited by John Grefenstette), pp. 224–230. Erlabum Associates, Hillsdale, NJ (1987).
15. D. Whitley, T. Starkweather and D. Fuquay, Scheduling problems and traveling salesman: the genetic edge recombination operator. *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 133–144. Morgan Kaufmann, Palo Slto, CA (1989).
16. J. H. Holland, *Adaption in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan (1975).
17. K. A. DeJong, Analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI (1975).
18. A. J. Vakharaia and U. Wemmerlow, Designing a cellular manufacturing system: a materials flow approach based on operations sequences. *IIE Transactions* **22**, 1, 84–97 (1990).